PUBLIC

Code Assessment

of the XChain SSR Oracle Smart Contracts

September 11, 2024

Produced for

SPARK

by

CHAINSECURITY

Contents

1 Executive Summary 3
2 Assessment Overview 5
3 Limitations and use of report 9
4 Terminology 10
5 Findings 11
6 Resolved Findings 12
7 Notes 14

1 Executive Summary

Dear all,

Thank you for trusting us to help SparkDAO with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of XChain SSR Oracle according to Scope to support you in forming an opinion on their security risks.

SparkDAO implements cross-chain oracles for the Sky Savings Rate where update messages are sent to L2s from Ethereum Mainnet.

The most critical subjects covered in our audit are functional correctness, access control and message passing.

The general subjects covered are code complexity and specification.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the XChain SSR Oracle repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	02 April 2024	01481b10aabb6b8a2c6afacec3cad19a90ecd7b1	Initial Version
2	05 April 2024	ccd09c8cf122200dc66affee7c07f8843cb4c684	After Intermediate Report
3	06 April 2024	463e012c51d50cd24d96fc5738e26f7ca624c94c	Added Base Support
4	24 April 2024	4361224e734be86e5666f49cd238ea183fa0902d	Added Arbitrum Support
5	03 July 2024	b868f15aca6b69d19f2f3005aa435cba9e5aee40	Architecture Changes and Zero max DSR mode
6	30 July 2024	2228ded849f963dbe90153d6c47ec0513a3513a1	Additional Event & README changes
7	07 September 2024	8f98580c3c48fdfcc64f65078e04d7d70ec291d4	Renaming and minor changes
8	11 September 2024	1938a3dffbb48f34b167541bf35c85b95fba61d4	Rename pot to sUSDS

For the solidity smart contracts, the compiler version `0.8.20` was chosen.

The following files were in scope:

```
src/DSRAuthOracle.sol
src/receivers/DSROracleReceiverOptimism.sol
src/receivers/DSROracleReceiverGnosis.sol
src/adapters/DSRBalancerRateProviderAdapter.sol
src/forwarders/DSROracleForwarderGnosis.sol
src/forwarders/DSROracleForwarderBase.sol
src/forwarders/DSROracleForwarderOptimism.sol
src/forwarders/DSROracleForwarder.sol
src/DSRMainnetOracle.sol
src/DSROracleBase.sol
```

In Version 4, the following contracts were added to the scope:

```
src/receivers/DSROracleReceiverArbitrum.sol
src/forwarders/DSROracleForwarderArbitrumOne.sol
```

Note that the following renaming occurred in Version 4:

```
src/forwarders/DSROracleForwarder.sol -> src/forwarders/DSROracleForwarderBase.sol
src/forwarders/DSROracleForwarderBase.sol -> src/forwarders/DSROracleForwarderBaseChain.sol
```

In version 5, the following contracts were removed:

```
src/receivers/DSROracleReceiverOptimism.sol
src/receivers/DSROracleReceiverGnosis.sol
src/receivers/DSROracleReceiverArbitrum.sol
```

In version 7, the following renamings occurred:

```
src/DSRAuthOracle.sol -> src/SSRAuthOracle.sol
src/adapters/DSRBalancerRateProviderAdapter.sol -> src/adapters/SSRBalancerRateProviderAdapter.sol
src/forwarders/DSROracleForwarderGnosis.sol -> src/forwarders/SSROracleForwarderGnosis.sol
src/forwarders/DSROracleForwarderOptimism.sol -> src/forwarders/SSROracleForwarderOptimism.sol
src/forwarders/DSROracleForwarderBase.sol -> src/forwarders/SSROracleForwarderBase.sol
src/DSRMainnetOracle.sol -> src/SSRMainnetOracle.sol
src/DSROracleBase.sol -> src/SSROracleBase.sol
src/forwarders/DSROracleForwarderArbitrumOne.sol -> src/forwarders/SSROracleForwarderArbitrum.sol
```

In version 7, the following contracts were removed:

```
src/forwarders/DSROracleForwarderBaseChain.sol
```

2.1.1 Excluded from scope

All other files including tests and helpers. The correctness of the external systems (i.e. bridges) is out of scope. Of the xchain-helpers, only the functions used have been reviewed.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

SparkDAO offers a framework to report values related to the DAI Savings Rate (DSR) from Ethereum Mainnet to various chains. The set of contracts consists of forwarders on L1 (Ethereum Mainnet) and receivers on L2. Currently, Gnosis, Optimism and Base are supported L2s. Oracle contracts are provided for users to access these cached values.

2.2.1 Oracles

DSROracleMainnetOracle

This oracle is deployed on mainnet and pulls its data directly from the Pot, the DAI Savings Rate contract. The values cached locally can be updated permissionlessly using the function `refresh()`.

DSRAuthOracle

Authenticated oracle which receives updates from Mainnet through the bridge. Exposes `setPotData()` for the cross-chain message receiver to update the data. The oracle only accepts increasing `rho` values that are in the past, DSR rates greater than one and less than the upper bound `maxDSR` which is defined by governance with `setMaxDSR()`, and increasing `chi` values that are upper bounded by a maximum chi that could have occurred assuming that the DSR never exceeded its maximum. Note that a `maxDSR`

being zero implements a special case where no maximum for the DSR is set so that no maximum value checks on the DSR and `chi` are applied.

All Oracle contracts, through inheritance of DSROracleBase, expose the following public functionality to access the data:

- `getPotData()`: Returns the struct `PotData` which consists of: `uint96 dsr`, the DAI Savings Rate in per second value (in the unit of `ray`), `uint120 chi`, the last computed conversion rate (to DAI) in `ray` and `rho` the timestamp of the last update (computation of `chi`) in seconds.
- `getDSR()`: returns the current DSR value stored.
- `getChi()`: returns the current Chi value stored.
- `getRho()`: returns the current Rho value stored.
- `getAPR()`: calculates and returns the Annual Percentage Rate calculated using the stored DSR.
- `getConversionRate()`: returns the conversion rate at the current timestamp.
- `getConversionRate(uint256 timestamp)`: returns the conversion rate at the given timestamp (now or in the future).
- `getConversionRateBinomialApprox()`: returns the binomial approximated conversion rate at the current timestamp.
- `getConversionRateBinomialApprox(uint256 timestamp)`: returns the binomial approximated conversion rate that the given timestamp (now or in the future)
- `getConversionRateLinearApprox()`: returns the linear approximated conversion rate at the current timestamp.
- `getConversionRateLinearApprox(uint256 timestamp)`: returns the linear approximated conversion rate at the given timestamp (now or in the future).

All values returned are based on the cached values which might be outdated. Updates must be triggered when the DSR changes, in between updates of the DSR the current `chi` can be calculated accurately based on the cached data. Calculating values for timestamps in the future may be inaccurate if the DSR changes.

2.2.2 Bridging framework

To publish data to the authorized oracle, cross-chain messages are sent from Ethereum mainnet to the supported chains. The implemented contracts leverage the xchain-helpers library (its review is part of another report).

Forwarder contracts are implemented to send messages while Receiver contracts are implemented to receive messages so that they can call the corresponding `DSRAuthOracle`.

Forwarders on Mainnet

For each supported chain, a corresponding Forwarder exists. Each exposes `refresh()`, allowing any caller to push current data to the respective chain.

Each forwarder leverages the corresponding Forwarder library from `xchain-helpers` to perform sending the message. More specifically, the following setup is expected:

1. Gnosis: `DSROracleForwarderGnosis` uses `AMBForwarder.sendMessageEthereumToGnosisChain`
2. Optimism: `DSROracleForwarderOptimism` uses `OptimismForwarder.sendMessageL1toL2` (with the Optimism cross-domain messenger)
3. Base: `DSROracleForwarderBaseChain` uses `OptimismForwarder.sendMessageL1toL2` (with the Base cross-domain messenger)
4. Arbitrum: `DSROracleForwarderArbitrumOne` uses `ArbitrumForwarder.sendMessageL1toL2` (note any overhead gas will be burned)

All of them inherit from the base contract DSROracleForwarderBase which handles message packing, stores the last seen Pot data and emits a corresponding event. It implements functions `getLastSeenPotData()`, `getLastSeenDSR()`, `getLastSeenChi()` and `getLastSeenRho()` to query the respective data. Note that the contract requires that the

Receivers on L2

While the receiver remains undefined in the repository in scope, the receivers intended for use are the "native" receivers defined in `xchain-helpers`. These perform bridge-specific access control for the received message and forward the call to a target contract. In the context of XChain SSR Oracle, it is expected that the target contract is the authorized oracle. Further, it is expected that the following contracts a deployed on the respective chains:

1. Gnosis: `AMBReceiver`
2. Optimism: `OptimismReceiver`
3. Base: `OptimismReceiver`
4. Arbitrum: `ArbitrumReceiver`

2.2.3 Adapter

Furthermore, DSRBalancerRateProviderAdapter is provided which exposes `getRate()`, a function returning the value of sDAI in terms of DAI.

2.2.4 Changes in Version 7

In version 7, the codebase has been adapted to work with the new Sky Savings Rate (SSR) instead of the the Dai Savings Rate (DSR). Ultimately, the codebase is now intended to be used with sUSDS instead of the Pot and the sDAI contract.

Additionally, the Arbitrum and Optimism forwarders' constructors have been generalized to receive the cross-domain contract as an argument (e.g. same contract can be used for Optimism and Base now, however, with different constructor arguments).

2.2.5 Changes in Version 8

In version 8, `pot` related namings have been updated to sUSDS.

2.3 Trust Model & Roles

- Pot: DAI Savings Rate contract. Fully trusted, source of Data. As of version 7, this contract corresponds to the sUSDS contract.
- Bridges: Fully trusted. Transmit the message, they must not alter any data nor censor any message. However, note that there are some limitations put in place to reduce the trust in bridges.
- Governance: Fully trusted. The governance can replace the callers of `setPotData()` which could lead to oracle manipulations. As of version 8, this function has been renamed to `setSUSDSData()`.

Receiver contracts update `DSRAuthOracle` based on received messages, which must originate from the designated contract on the L1 chain. At the deployment of these receiver contracts, the messaging contract and L1 origin parameters are initialized, and the correct configuration is essential.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0
Informational Findings	3

- Outdated IPot Interface Code Corrected
- Redundant Getters Code Corrected
- Inconsistencies Code Corrected

6.1 Outdated IPot Interface

Informational Version 7 Code Corrected

CS-XDSR-003

The initial version of the system is expected to work with sDAI, hence the `dss.pot` contract is used, where the state variables `dsr`, `chi`, and `rho` are of type `uint256`. After the rebranding, the new oracle will integrate with sUSDS, where `ssr` is `uint256` but `chi` is `uint192` and rho is `uint64`. However, the interface is not updated and still expects `uint256`.

Code corrected:

The outdated `IPot.sol` interface has been replaced by `ISUSDS.sol`.

6.2 Inconsistencies

Informational Version 1 Code Corrected

CS-XDSR-002

The codebase is inconsistent in style across files and functions. Below is an incomplete list:

1. The DSRAuthOracle explicitly uses `1e27`. However, in other places, the ray constant RAY is used.
2. The `getConversionRate` function does not perform the time-delta computation `timestamp - rho` in an `unchecked` block while `getConversionRateLinearApprox()` and `getConversionRateBinomialApprox()` do.

Code corrected:

The code has been changed accordingly.

6.3 Redundant Getters

Informational | Version 1 | Code Corrected

CS-XDSR-001

`_lastSeenPotData` is a `public` variable and thus has an autogenerated public getter. Nevertheless, there is a second getter `getLastSeenPotData()`. Ultimately, both getters return the same since no complex data types are used in `PotData`.

Code corrected:

`_lastSeenPotData` is now `private`.

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

7.1 Delayed Oracle Consequences

Note Version 1

Users of the crosschain DSR oracle should be aware that the oracles could hold outdated values. They should ensure that the oracle is kept alive when changes in the DSR occur. Otherwise, the impact of the scenarios below could be more impactful to their application than expected.

Namely, the following scenarios could occur:

1. The DSR decreased on Mainnet. However, an L2 oracle has not been updated. Consequently, the L2 oracle will overvalue `chi` and thus will overvalue sDAI.
2. The DSR increased on Mainnet. However, an L2 oracle has not been updated. Consequently, the L2 oracle will undervalue `chi` and thus will undervalue sDAI.

7.2 Oracle Functions May Behave Differently

Note Version 1

Users and integrators of the DSR oracles should be aware that DSR functions `getConversionRate()`, `getConversionRateLinearApprox()` and `getConversionRateBinomialApprox()` do not differ only in accuracy but could also differ in terms of reverting behaviour of overflows. For example, there is a set of numbers for which the third function would revert values whereas the second would not.